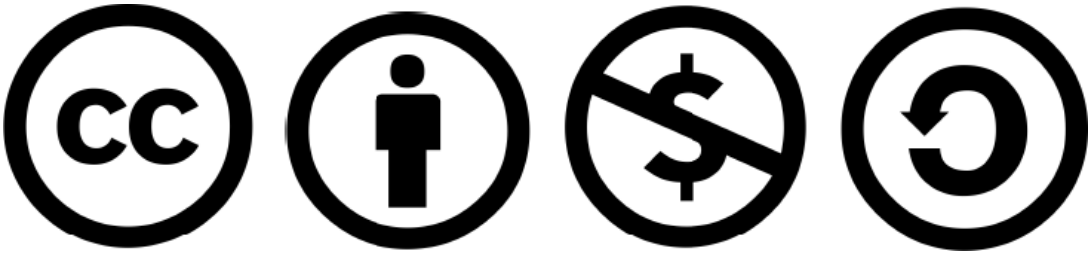
Report Contest-CUDA

Radix Sort



Course: High Performance Computing 2021/2022
Lecturer: Francesco Moscato - fmoscato@unisa.it

Group:
Lamberti Martina 0622701476 m.lamberti61@studenti.unisa.it
Salvati Vincenzo 0622701550 v.salvati10@studenti.unisa.it
Silvitelli Daniele 0622701504 d.silvitelli@studenti.unisa.it
Sorrentino Alex 0622701480 a.sorrentino120@studenti.unisa.it

Summary

Problem Description

Parallelizing and evaluating performances of "RADIX SORT" Algorithm, by using CUDA with different memory allocation:

1. Global Memory;
2. Shared Memory;
3. Texture Memory.

Experimental setup

Hardware

GPU

The scripts "***info.cu***" and "***bandwidth.cu***" reports information about GPU's configuration.

```
Device name                                : NVIDIA GeForce RTX 2060 SUPER
Compute capability                         : 7.5

Clock Rate                                 : 1695000 kHz
Total SMs                                  : 34
Shared Memory Per SM                       : 65536 bytes
Registers Per SM                           : 65536 32-bit
Max threads per SM                         : 1024
L2 Cache Size                              : 4194304 bytes
Total Global Memory                        : 8589606912 bytes
Memory Clock Rate                          : 7001000 kHz

Max threads per block                      : 1024
Max threads in X-dimension of block        : 1024
Max threads in Y-dimension of block        : 1024
Max threads in Z-dimension of block        : 64

Max blocks in X-dimension of grid          : 2147483647
Max blocks in Y-dimension of grid          : 65535
Max blocks in Z-dimension of grid          : 65535

Shared Memory Per Block                    : 49152 bytes
Registers Per Block                        : 65536 32-bit
Warp size                                  : 32

Pageable transfers (16MB)
  Host to Device bandwidth (GB/s)          : 6.995637
  Device to Host bandwidth (GB/s)          : 9.796663

Pinned transfers (16MB)
  Host to Device bandwidth (GB/s)          : 12.654791
  Device to Host bandwidth (GB/s)          : 12.685717
```

The NVIDIA GeForce RTX 2060 SUPER has cc=7.x, so that, the maximum number of manageable blocks for each SM is 16.

CPU

The command '***cat /proc/cpuinfo***' reports information about core's configuration.

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 0
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 0
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 2
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
```

```
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 1
cpu cores       : 10
apicid          : 0
initial apicid : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 1
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 4
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 2
cpu cores       : 10
```

```
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 5
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 2
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 6
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 3
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
```

```
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 7
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 3
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 8
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 4
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 9
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 4
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 10
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 5
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 11
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
```

```
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 5
cpu cores       : 10
apicid          : 0
initial apicid : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 12
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 6
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 13
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 6
cpu cores       : 10
apicid          : 0
```

```
initial apicid : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 14
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 7
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 15
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 7
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
```

```
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 16
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 8
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 17
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 8
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 18
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 9
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

```
processor       : 19
vendor_id       : GenuineIntel
cpu family      : 6
model           : 165
model name      : Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
stepping        : 5
microcode       : 0xffffffff
cpu MHz         : 3701.000
cache size      : 256 KB
physical id     : 0
siblings        : 20
core id         : 9
cpu cores       : 10
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 6
wp              : yes
flags            : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq
dtes64 est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes
xsave osxsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt ibrs ibpb stibp ssbd
bogomips        : 7402.00
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

RAM

The command '***cat /proc/meminfo***' gives information about total amount of physical RAM, the one left unused….

```
MemTotal:       16693720 kB
MemFree:         8456180 kB
Buffers:           34032 kB
Cached:           188576 kB
SwapCached:            0 kB
Active:           167556 kB
Inactive:         157876 kB
Active(anon):     103104 kB
Inactive(anon):    17440 kB
Active(file):      64452 kB
Inactive(file):   140436 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:      50331648 kB
SwapFree:       50221984 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        102824 kB
Mapped:            71404 kB
Shmem:             17720 kB
Slab:              13868 kB
SReclaimable:       6744 kB
SUnreclaim:         7124 kB
KernelStack:        2848 kB
PageTables:         2524 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:      515524 kB
Committed_AS:    3450064 kB
VmallocTotal:     122880 kB
VmallocUsed:       21296 kB
VmallocChunk:      66044 kB
HardwareCorrupted:     0 kB
AnonHugePages:      2048 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       12280 kB
DirectMap4M:      897024 kB
```

Software

- SO: Windows 11.
- The command "***nvcc –-version***" reports CUDA libraries.

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Nov_30_19:15:10_Pacific_Standard_Time_2020
Cuda compilation tools, release 11.2, V11.2.67
Build cuda_11.2.r11.2/compiler.29373293_0
```

- The command "***nvidia-smi***" reports monitoring and management capabilities.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 511.23       Driver Version: 511.23       CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ... WDDM  | 00000000:01:00.0  On |                  N/A |
|  0%   59C    P0    41W / 175W |   1120MiB /  8192MiB |      5%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Performance

In our implementation, the array to be sorted is read from the file "*random_numbers.txt*", because we wanted to generate the random numbers once, so that it is always used the same numbers to make comparisons.
Hence, to generate this file it was implemented a script called "*main.java*" placed in the folder which is called "*Random_Int_2000000_Saving_File*".

In particular, our algorithm is able to execute automatically the same code by following parameters:

- size_of_array: 5120;
- thread_per_block: 64, 128, 256, 512, 1024.

The choice of this threads is not random, since the GPU's SM is able to deal with max 16 blocks:

- $\frac{1024}{64} = 16\ blocks$ → 16 = 16 → $16 * 64 = 1024$ → the occupancy is 100%;
- $\frac{1024}{128} = 8\ blocks$ → 8 = 16 → $16 * 128 = 1024$ → the occupancy is 100%;
- $\frac{1024}{256} = 4\ blocks$ → 4 = 16 → $16 * 256 = 1024$ → the occupancy is 100%;
- $\frac{1024}{512} = 2\ blocks$ → 2 = 16 → $16 * 512 = 1024$ → the occupancy is 100%;
- $\frac{1024}{1024} = 1\ blocks$ → 1 = 16 → $16 * 1024 = 1024$ → the occupancy is 100%;

As far as this program concerned, it takes into account about the elapsed time and the Mflops referred R*adix Sort algorithm* performed by CPU and GPU.

The bash file allows us to perform this automation 200 times in order to calculate a mean of each measure.

To perform the number of flops, it has been considered all ALU operations from each implemented kernel. In addition to that, each number of them has been multiplied for the number of elements (each element is associated to each active thread, since the expected occupancy is always 100%) and for 32 (because of the considered binary digit).

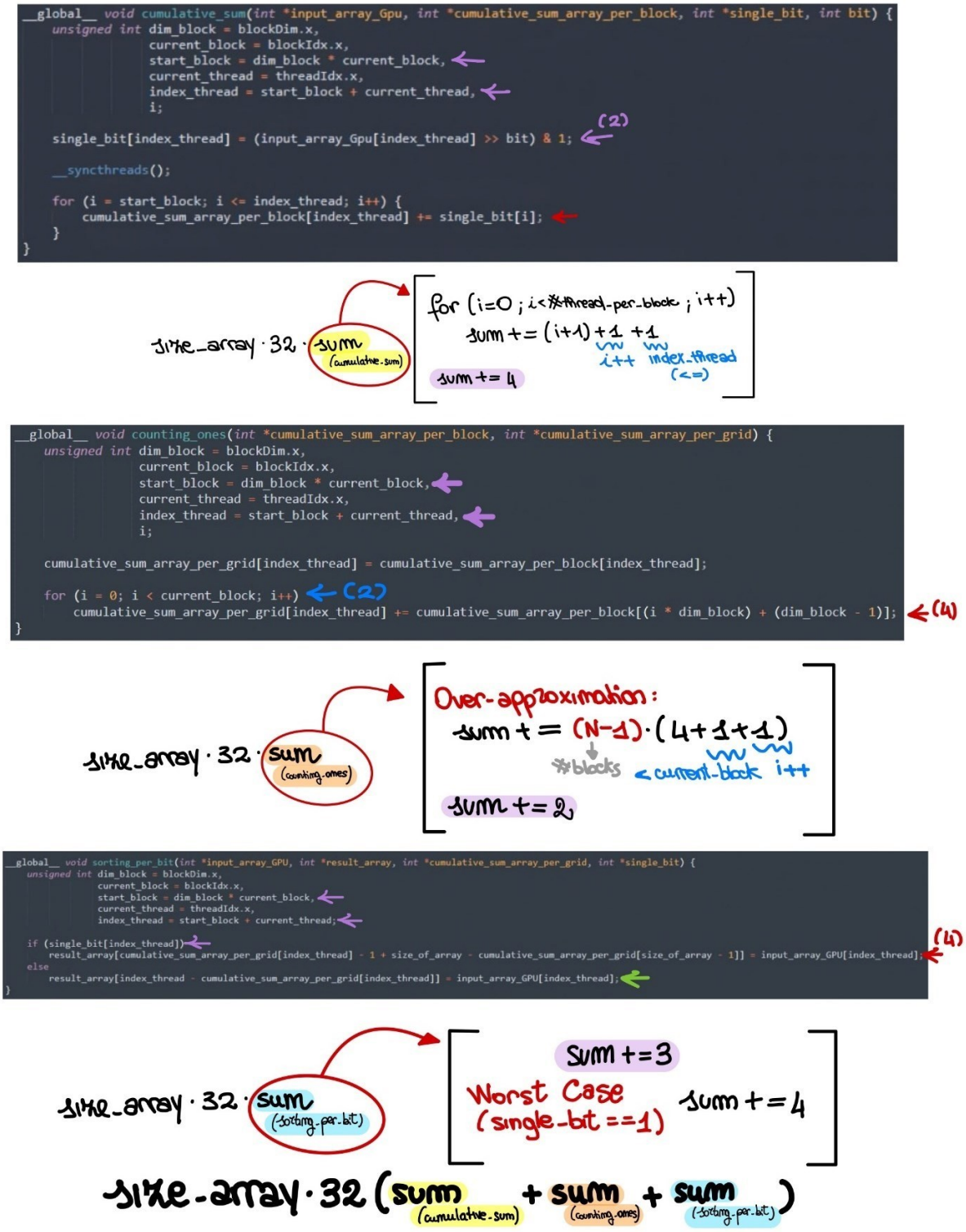


Figure 1 - counting flops

Global Memory

THREADS PER BLOCK = 64	GLOBAL MEMORY
Mflops/s	427357,55765625
Elapsed time GPU	0,00104405
Elapsed time CPU	0,00384000
THREADS PER BLOCK = 128	GLOBAL MEMORY
Mflops/s	1350564,55125000
Elapsed time GPU	0,00106650
Elapsed time CPU	0,00375000
THREADS PER BLOCK = 256	GLOBAL MEMORY
Mflops/s	1137445,01031250
Elapsed time GPU	0,00105993
Elapsed time CPU	0,00376000
THREADS PER BLOCK = 512	GLOBAL MEMORY
Mflops/s	183299,90839844
Elapsed time GPU	0,00121245
Elapsed time CPU	0,00357500
THREADS PER BLOCK = 1024	GLOBAL MEMORY
Mflops/s	174185,31914063
Elapsed time GPU	0,00244899
Elapsed time CPU	0,00386500

Figure 2 - measures of global memory

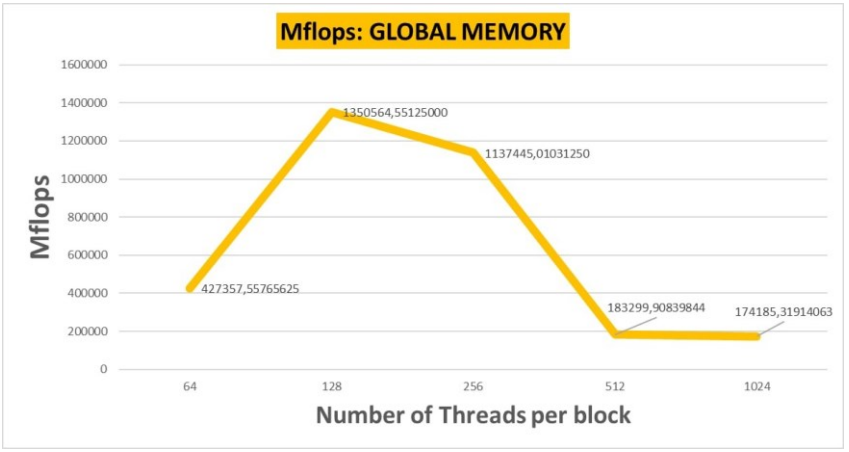


Figure 3 - Mflops of global memory

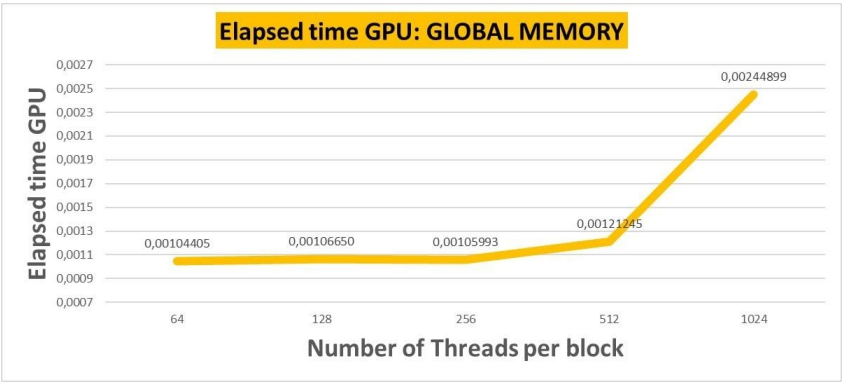


Figure 4 - Elapsed time GPU of global memory

Shared Memory

THREADS PER BLOCK = 64	SHARED MEMORY
Mflops/s	429702,49695313
Elapsed time GPU	0,00103509
Elapsed time CPU	0,00386500
THREADS PER BLOCK = 128	SHARED MEMORY
Mflops/s	1410981,33437500
Elapsed time GPU	0,00102128
Elapsed time CPU	0,00385500
THREADS PER BLOCK = 256	SHARED MEMORY
Mflops/s	1203183,41343750
Elapsed time GPU	0,00100178
Elapsed time CPU	0,00367500
THREADS PER BLOCK = 512	SHARED MEMORY
Mflops/s	222608,88476563
Elapsed time GPU	0,00099867
Elapsed time CPU	0,00370000
THREADS PER BLOCK = 1024	SHARED MEMORY
Mflops/s	346588,60687500
Elapsed time GPU	0,00123352
Elapsed time CPU	0,00380500

Figure 5 - measures of shared memory

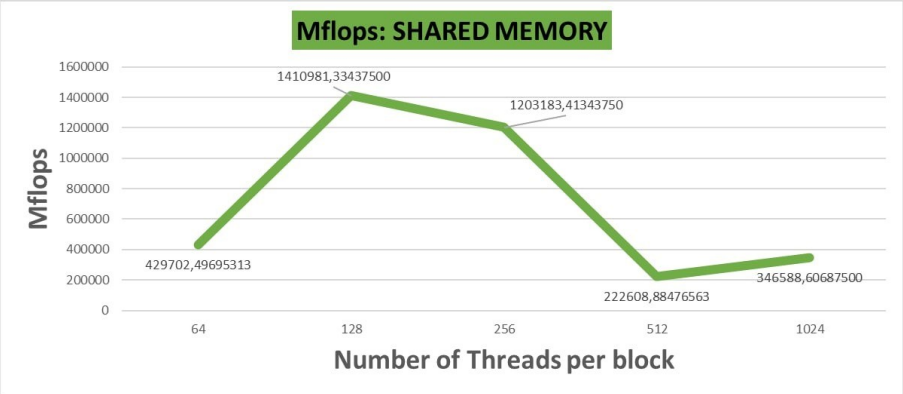


Figure 6 - Mflops of shared memory

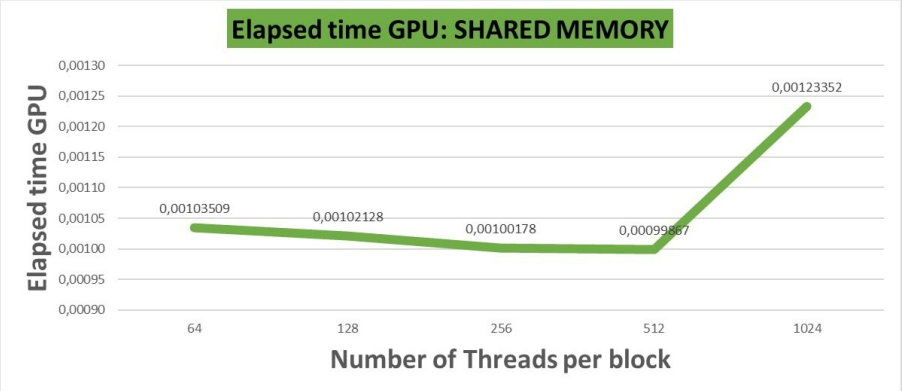


Figure 7 - Elapsed time GPU of shared memory

Texture Memory

THREADS PER BLOCK = 64	TEXTURE MEMORY
Mflops/s	395931,18171875
Elapsed time GPU	0,00111952
Elapsed time CPU	0,00379500
THREADS PER BLOCK = 128	TEXTURE MEMORY
Mflops/s	1280611,88687500
Elapsed time GPU	0,00112348
Elapsed time CPU	0,00367000
THREADS PER BLOCK = 256	TEXTURE MEMORY
Mflops/s	1081408,79218750
Elapsed time GPU	0,00111389
Elapsed time CPU	0,00362000
THREADS PER BLOCK = 512	TEXTURE MEMORY
Mflops/s	201442,80390625
Elapsed time GPU	0,00110267
Elapsed time CPU	0,00386500
THREADS PER BLOCK = 1024	TEXTURE MEMORY
Mflops/s	318997,21328125
Elapsed time GPU	0,00133933
Elapsed time CPU	0,00386500

Figure 8 - measures of texture memory

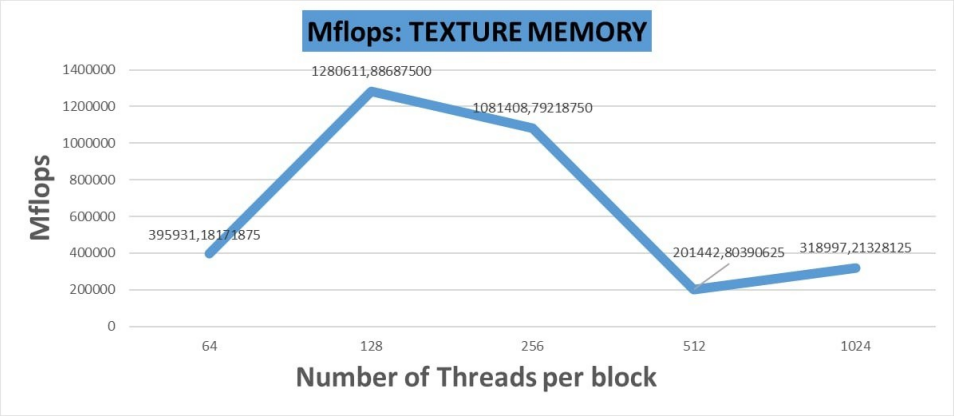


Figure 9 - Mflops of texture memory

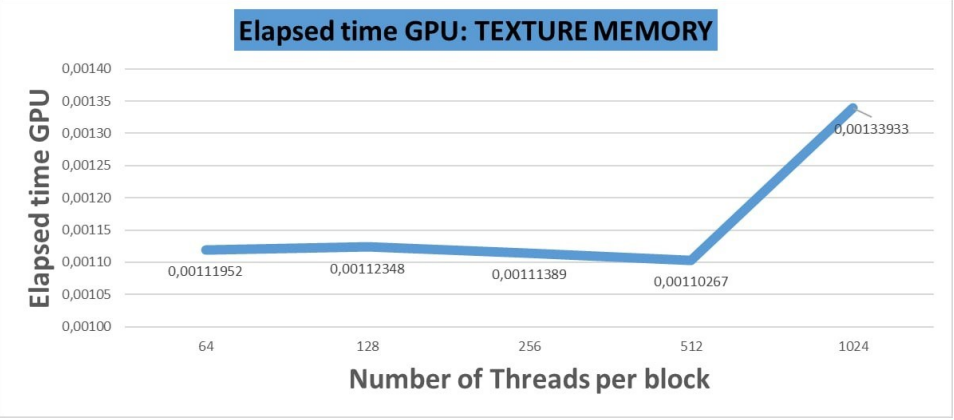


Figure 10 - Elapsed time GPU of texture memory

Considerations

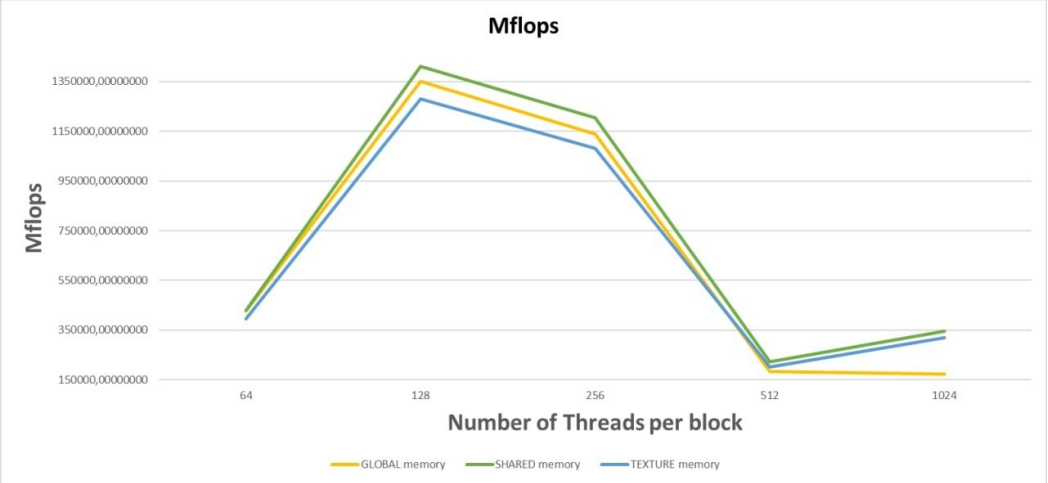


Figure 11 - Mflops of each memory allocation

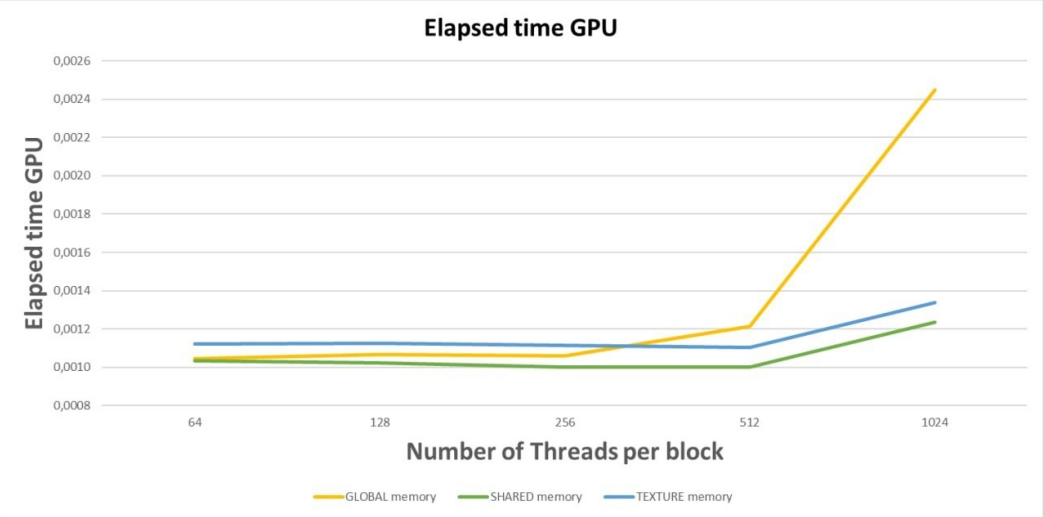


Figure 12 - Elapsed time GPU of each memory allocation

The plots take into account about the Mflops and the elapsed time for each different memory allocation. As follow, it will be explained three particular considerations:

- **Global memory has high latency due to technological reasons**: global memory has the worst performance than others, because of the presence of the L2 cache. In fact, every access pass through the cache, so that it could have cache misses;
- **The profitable way of performing computation on the device is to use shared memory**: shared memory has better performance than others, despite a necessary overhead to load subsets of data from global memory. In fact, it has lower latency than others and this compensates for the overhead;
- **Texture memory is a read-only cache ad has high latency due to technological reasons**: texture memory has better performance than global memory for high number of threads, anyway, as global memory, it could have cache misses.

Case Study

We have produced solutions with the same algorithm, but they differ in the way the memory is allocated.

The kernel is divided into blocks of dimension $\frac{size\_of\_array}{thread\_per\_block}$ and each thread carry on specific element of input array.

So, the implementation operates with unsigned int numbers since the Radix Sort works only with positive numbers, moreover, they are considered in binary numerical system ***(that led us to represent decimal numbers on 32 bits)***. Hence, we iterate over all the bits of each number from the LSB to MSB.

To better use the GPU, three kernels have been used. The reason is related to the synchronization of all threads of the grid and not only the ones in a single block. In this way, the operations within the single kernel will be executed only after the conclusion of previous kernel operations.

```
// Run kernels
for (bit = 0; bit < 32; bit++) {
    // Reset "cumulative_sum_array_per_block"
    check_error_from(cudaMemset(cumulative_sum_array_per_block, 0, size_of_array_in_bytes));

    cumulative_sum_per_block<<<dim_grid, dim_block>>>(input_array_GPU,
                                                      cumulative_sum_array_per_block,
                                                      single_bit,
                                                      bit);
    check_error_from(cudaGetLastError());

    counting_ones_per_grid<<<dim_grid, dim_block>>>(cumulative_sum_array_per_block,
                                                    cumulative_sum_array_per_grid);
    check_error_from(cudaGetLastError());

    sorting_per_bit<<<dim_grid, dim_block>>>(input_array_GPU,
                                             result_array,
                                             cumulative_sum_array_per_grid,
                                             single_bit);
    check_error_from(cudaGetLastError());

    // Set new "input_array_GPU" by sub ordered array "result_array"
    check_error_from(cudaMemcpy(input_array_GPU, result_array, size_of_array_in_bytes, cudaMemcpyDeviceToDevice));
}
```

Figure 13 - Kernels

The iteration of this loop led us to consider a specific bit of the decimal numbers.

1. CUMULATIVE_SUM_PER_BLOCK

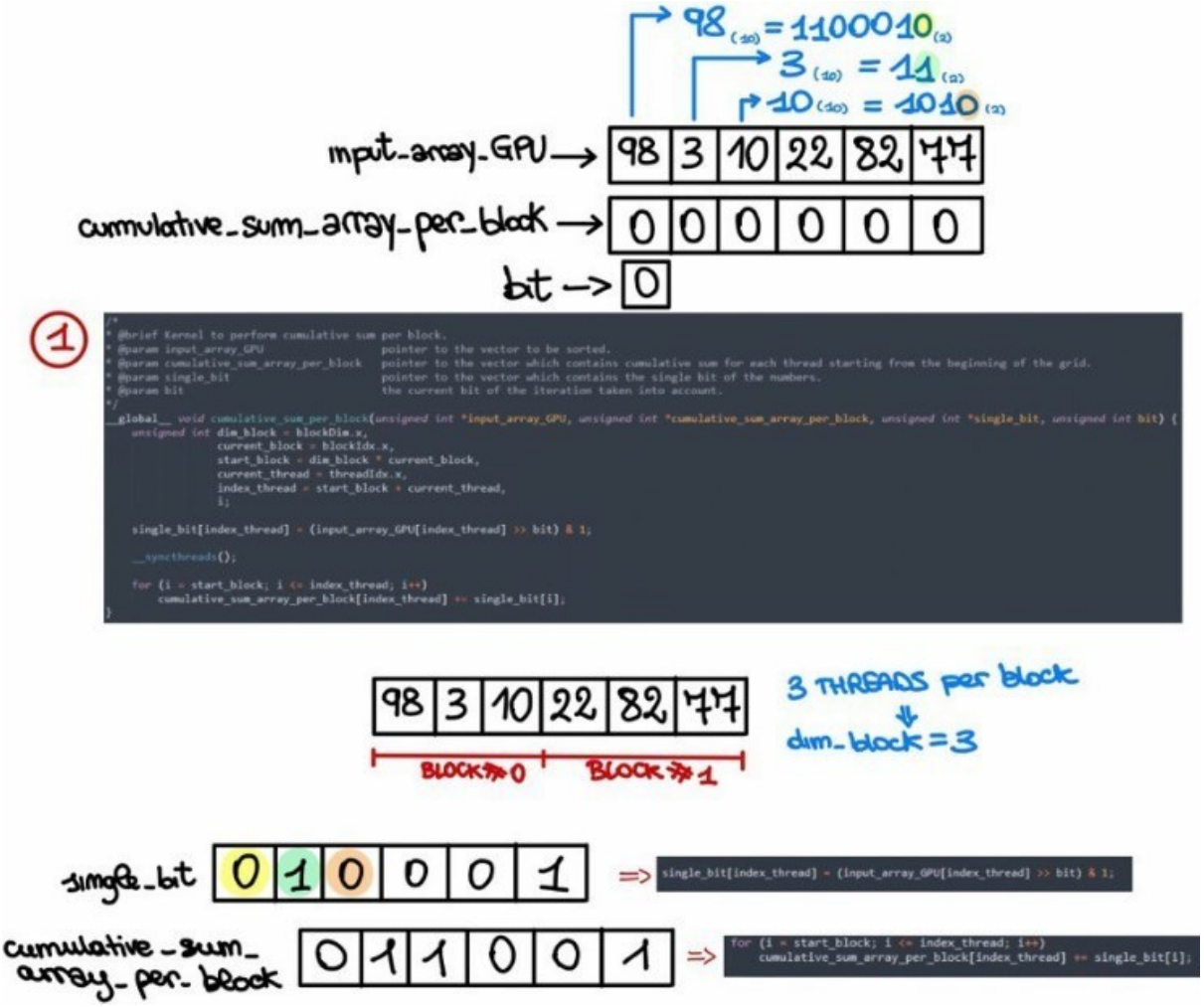


Figure 14 - 1° Kernel

In this kernel it is considered `single_bit` array which contains the i-th bit of the `index_thread-th` decimal number stored into `input_array_GPU`.

The i-th bit of this array is obtained by two operations:

1. a shift operation;
2. and logic AND operation.

As each thread need to know the binary elements considered in own block, it is placed *__syncthreads()* function and, at this point, each thread performs the sum of each elementuntil itself, limiting to its own block.
Thus, each performed cumulative sum is stored into `cumulative_sum_array_per_block`.

At the end of this kernel, we have filled all `cumulative_sum_array_per_block` which can be properly used in the next kernel.

2. COUNTING ONES PER GRID

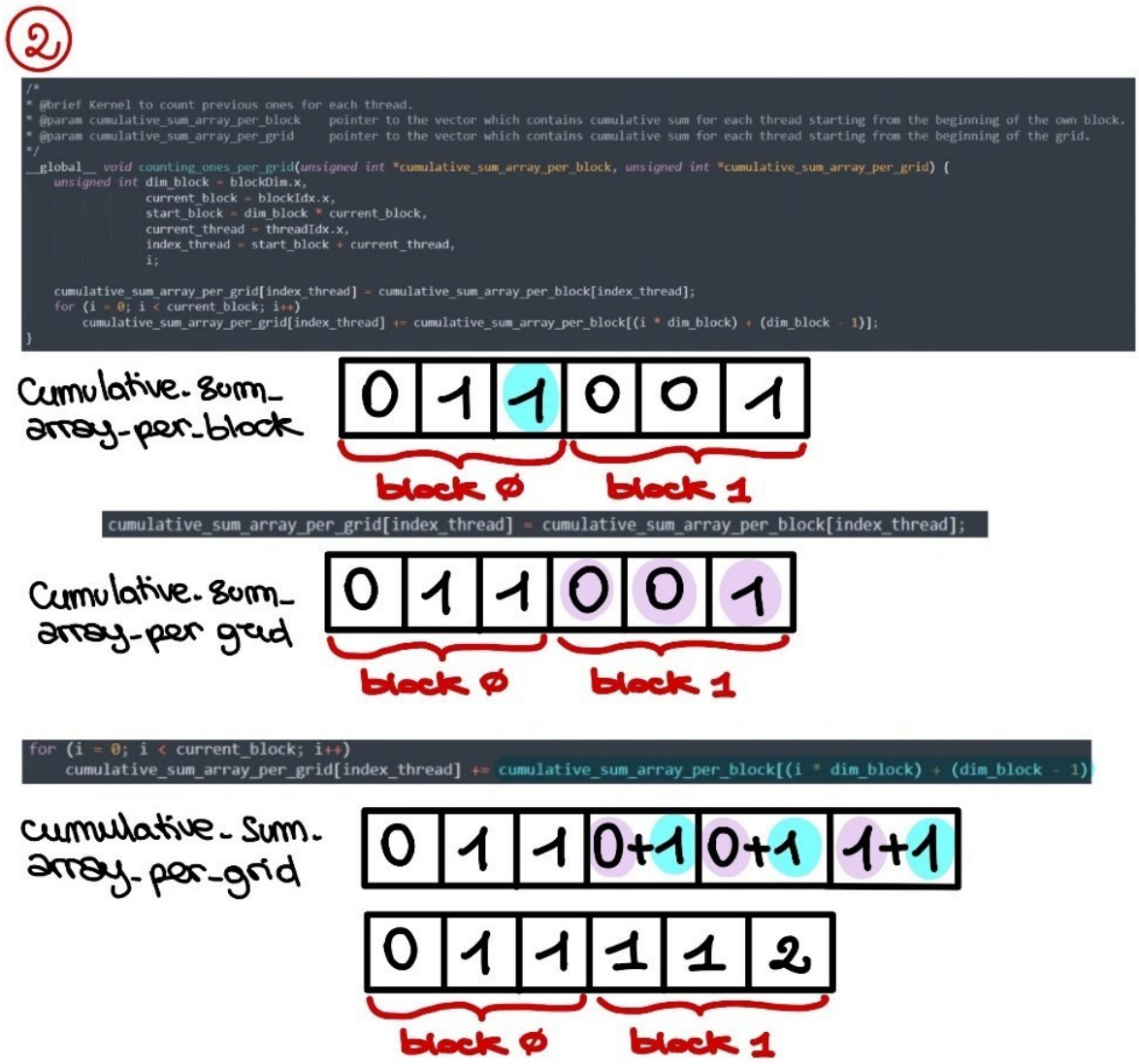


Figure 15 - 2° Kernel

Given a thread of a specific block, the goal of this kernel is to compute a cumulative sum obtained considering the cumulative sum per block performed by each last thread of each previous block plus the cumulative sum per block performed by itself in its own block.

3. SORTING PER BIT

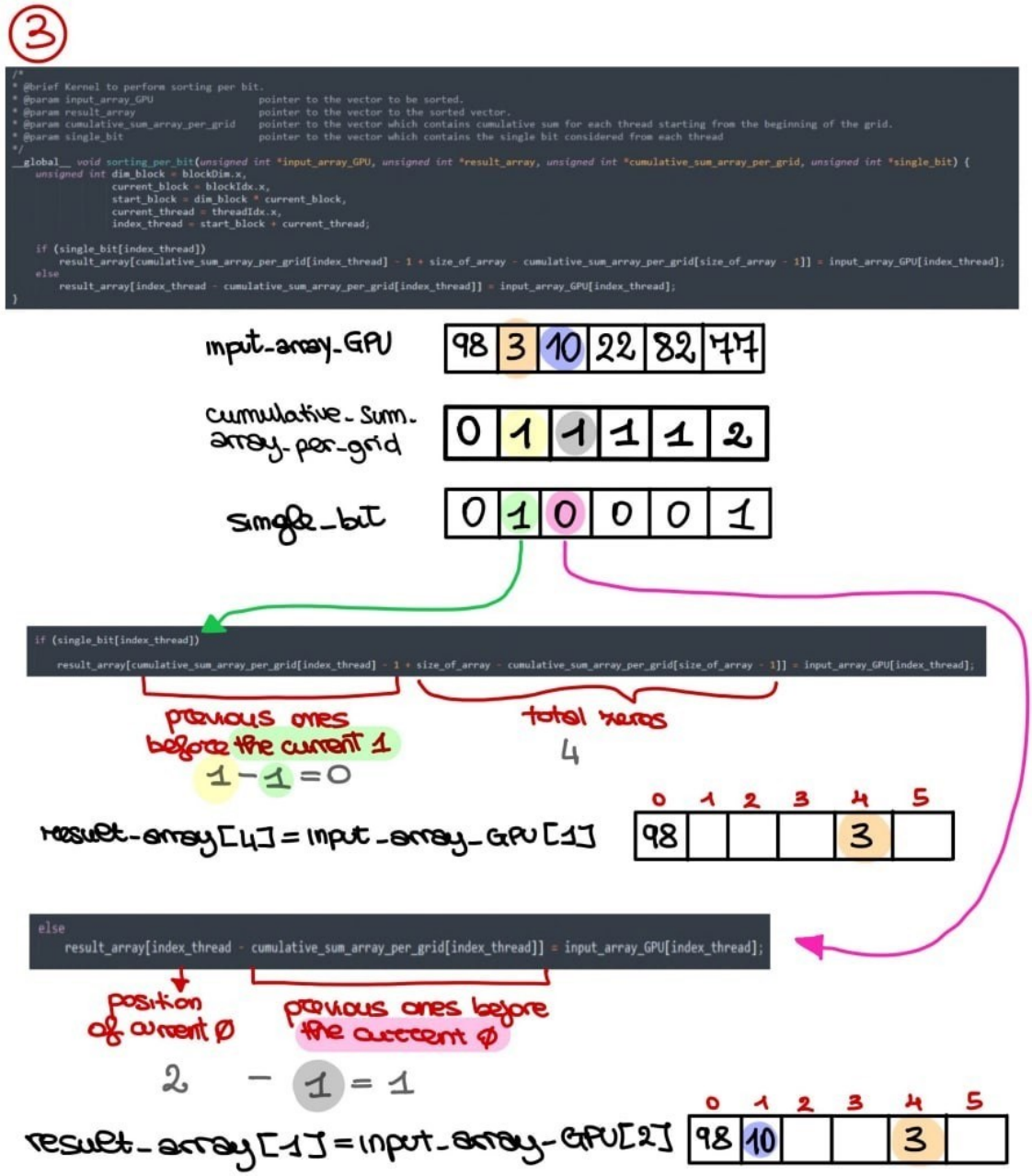


Figure 16 - 3° Kernel

Finally, the last kernel is used to sort the decimal element by using properly the binary digits and the performed cumulative sum. In particular, given a specific decimal number, the considered binary digit:

- if it is one, it must be placed in a particular position of `result_array` obtained by putting all binary zero digits ($size\_of\_array - cumulative\_sum\_array\_per\_grid[size\_of\_array - 1]$) and all previous binary one digits ($cumulative\_sum\_array\_per\_grid[index\_thread] - 1$) before it;
- if it is zero, it must be placed in a particular position of `result_array` obtained by subtracting the current position ($index\_thread$) to the number of all previous binary one digits ($cumulative\_sum\_array\_per\_grid[index\_thread]$) before it.

Differences among different memory allocation

As mentioned above, each version runs the same procedure, however specific arrays, used in the kernels, are allocated differently.

In particular, respect to the easy version that uses global memory, the shared memory is designed to optimize cyclic operations which require multiple memory accesses in order to justify the overhead of the data initialization from global to shared. In addition to that, the decision to use shared only for specific arrays is also linked to the fact that operations are limited to the same block; having a very limited amount of shared memory per block avoids saturation of the same as it is certain that the data stored does not fill it.

The use of *__syncthreads()* has been adopted to use a consistent shared memory.

```
__global__ void cumulative_sum_per_block(unsigned int *input_array_GPU, unsigned int *cumulative_sum_array_per_block, unsigned int *single_bit, unsigned int bit) {
    unsigned int dim_block = blockDim.x,
                 current_block = blockIdx.x,
                 start_block = dim_block * current_block,
                 current_thread = threadIdx.x,
                 index_thread = start_block + current_thread,
                 i;

    __shared__ unsigned int shared_single_bit[thread_per_block];
    __shared__ unsigned int shared_cumulative_sum_array_per_block[thread_per_block];

    shared_cumulative_sum_array_per_block[current_thread] = cumulative_sum_array_per_block[index_thread];
    shared_single_bit[current_thread] = ((input_array_GPU[index_thread] >> bit) & 1);

    __syncthreads();

    for (i = 0; i <= current_thread; i++)
        shared_cumulative_sum_array_per_block[current_thread] += shared_single_bit[i];

    single_bit[index_thread] = shared_single_bit[current_thread];
    cumulative_sum_array_per_block[index_thread] = shared_cumulative_sum_array_per_block[current_thread];
}
```

Figure 17 – use case of shared memory

Finally, the texture memory version followed a mixed approach, in that shared memory was also used, paying attention about read-only cache.

```
__global__ void cumulative_sum_per_block(unsigned int *input_array_GPU, unsigned int *cumulative_sum_array_per_block, unsigned int *single_bit, unsigned int bit) {
    unsigned int dim_block = blockDim.x,
                 current_block = blockIdx.x,
                 start_block = dim_block * current_block,
                 current_thread = threadIdx.x,
                 index_thread = start_block + current_thread,
                 i;

    __shared__ unsigned int shared_single_bit[thread_per_block];
    __shared__ unsigned int shared_cumulative_sum_array_per_block[thread_per_block];

    shared_cumulative_sum_array_per_block[current_thread] = get_cumulative_sum_array_per_block_texture_element(index_thread);
    shared_single_bit[current_thread] = ((get_input_array_GPU_texture_element(index_thread) >> bit) & 1);

    __syncthreads();

    for (i = 0; i <= current_thread; i++)
        shared_cumulative_sum_array_per_block[current_thread] += shared_single_bit[i];

    single_bit[index_thread] = shared_single_bit[current_thread];
    cumulative_sum_array_per_block[index_thread] = shared_cumulative_sum_array_per_block[current_thread];
}
```

Figure 18 – use case of texture memory

Test case

To check the correctness of the ordered array, it has been implemented a function which compare the sorted array produced by the CPU and the one produced by GPU:

- void ***compare_output_between_CPU_and_GPU*** (unsigned int *output_array_CPU, unsigned int *output_array_GPU)
  Test function to compare output between CPU and GPU.
  Parameters
  - output_array_CPU pointer to the sorted vector by CPU.
  - output_array_GPU pointer to the sorted vector by GPU.

Indeed, to determine the number wrong elements it has been implemented another function:

- unsigned int ***errors_in_comparison_between*** (unsigned int *output_array_CPU, unsigned int *output_array_GPU)
  Test function to detect errors between CPU and GPU.
  Parameters
  - output_array_CPU pointer to the sorted vector by CPU.
  - output_array_GPU pointer to the sorted vector by GPU.

API

Implemented functions

Type	Name	Used by
unsigned int	***get_max*** (unsigned int **array*) Function to get the largest element from an array.	***CPU***
void	***localsort*** (unsigned int **array*, unsigned int *place*) Function to execute localsort.	***CPU***
void	***radix_sort_on_CPU*** (unsigned int *input_array, unsigned int *output_array_CPU, double *elapsed_from_CPU) Function to perform RadixSort by using CPU.	***CPU***
unsigned int	***get_input_array_GPU_texture_element*** (unsigned int index) Device function to get element from texture reference.	***Texture***
unsigned int	***get_cumulative_sum_array_per_block_texture_element*** (unsigned int index) Device function to get element from texture reference.	***Texture***
unsigned int	***get_cumulative_sum_array_per_grid_texture_element*** (unsigned int index) Device function to get element from texture reference.	***Texture***
unsigned int	***get_single_bit_texture_element*** (unsigned int index) Device function to get element from texture reference.	***Texture***
void	***cumulative_sum_per_block*** (unsigned int *input_array_GPU, unsigned int *cumulative_sum_array_per_block, unsigned int *single_bit, unsigned int bit) Kernel to perform cumulative sum per block.	***Global Shared Texture***
void	***counting_ones_per_grid*** (unsigned int *cumulative_sum_array_per_block, unsigned int *cumulative_sum_array_per_grid) Kernel to count previous ones for each thread.	***Global Shared Texture***

void	***sorting_per_bit*** (unsigned int *input_array_GPU, unsigned int *result_array, unsigned int *cumulative_sum_array_per_grid, unsigned int *single_bit) Kernel to perform sorting per bit.	***Global Shared Texture***
float	***perform_milliseconds_to_seconds*** (float elapsed_in_ms) Function to perform time from milliseconds to seconds.	***Global Shared Texture***
unsigned int	***count_flops*** (unsigned int number_of_all_blocks) Function to count flops.	***Global Shared Texture***
float	***perform_Mflop_per_sec*** (float elapsed_in_ms, unsigned int number_of_all_blocks) Function to perform Mflops.	***Global Shared Texture***
void	***check_error_from*** (cudaError_t cuda_error, bool show_result = true) Function to check error from CUDA operation.	***Global Shared Texture***
void	***radix_sort_on_GPU*** (unsigned int *input_array, unsigned int *output_array_GPU, unsigned int *number_of_all_blocks, float *mflop_per_sec, float *elapsed_from_GPU) Function to perform RadixSort by using GPU.	***Global Shared Texture***
void	***read_input_from_file*** (unsigned int **array*) Function to read an unsorted array from a file.	***CPU***
bool	***file_exist*** (char *filename) Function to check if file exists.	***CPU***
void	***make_csv_of*** (unsigned int number_of_all_blocks, float mflop_per_sec, float elapsed_from_GPU, double elapsed_from_CPU) Function to make a csv of the measurements.	***CPU***
unsigned int	***errors_in_comparison_between*** (unsigned int *output_array_CPU, unsigned int *output_array_GPU) Test function to detect errors between CPU and GPU.	***CPU***

void	***compare_output_between_CPU_and_GPU*** (unsigned int *output_array_CPU, unsigned int *output_array_GPU) Test function to compare output between CPU and GPU.	***CPU***

Implemented Functions Documentation

- unsigned int ***get_max*** (unsigned int *array)
  Function to get the largest element from an array.
  Parameters
  - array pointer to the vector to be sorted.

- void ***localsort*** (unsigned int **array*, unsigned int *place*)
  Function to detect errors between CPU and GPU.
  Parameters
  - *array* pointer to the vector to be sorted.
  - *place* current digit which it is considered (units, tens, hundreds, ...).

- void ***radix_sort_on_CPU*** (unsigned int *input_array, unsigned int *output_array_CPU, double *elapsed_from_CPU)
  Function to compare output between CPU and GPU.
  Parameters
  - input_array pointer to the vector to be sorted.
  - output_array_GPU pointer to the sorted vector.
  - elapsed_from_CPU elapsed time committed from CPU

- __device__ unsigned int ***get_input_array_GPU_texture_element*** (unsigned int *index)
  Device function to get element from texture reference.
  Parameters
  - index of input_array_GPU_texture_reference.

- __device__ unsigned int ***get_cumulative_sum_array_per_block_texture_element*** (unsigned int *index)
  Device function to get element from texture reference.
  Parameters
  - index index of cumulative_sum_array_per_block_texture_reference.

- __device__ unsigned int ***get_cumulative_sum_array_per_grid_texture_element*** (unsigned int *index)
  Device function to get element from texture reference.
  Parameters
  - index index of cumulative_sum_array_per_grid_texture_reference.

- __device__ unsigned int ***get_single_bit_texture_element*** (unsigned int *index)Device function to get element from texture reference.

  Parameters

  - index index of single_bit_texture_reference.

- __global__ void **cumulative_sum_per_block** (unsigned int *input_array_GPU, unsigned int *cumulative_sum_array_per_block, unsigned int *single_bit, unsigned int bit)
  Kernel to perform cumulative sum per block.

  Parameters

  - input_array_GPU pointer to the vector to be sorted.
  - cumulative_sum_array_per_block pointer to the vector which contains cumulative sum for each thread starting from the beginning of the own block.
  - single_bit pointer to the vector which contains the single bit of the numbers.
  - bit the current bit of the iteration taking into account.

- __global__ void **counting_ones_per_grid** (unsigned int *cumulative_sum_array_per_block, unsigned int *cumulative_sum_array_per_grid)
  Kernel to count previous ones for each thread.

  Parameters

  - cumulative_sum_array_per_block pointer to the vector which contains cumulative sum for each thread starting from the beginning of the own block.
  - cumulative_sum_array_per_grid pointer to the vector which contains cumulative sum for each thread starting from the beginning of the grid.

- __global__ void **sorting_per_bit** (unsigned int *input_array_GPU, unsigned int *result_array, unsigned int *cumulative_sum_array_per_grid, unsigned int *single_bit)
  Kernel to perform sorting per bit.

  Parameters

  - input_array_GPU pointer to the vector to be sorted.
  - result_array pointer to the vector to the sorted vector.
  - cumulative_sum_array_per_grid pointer to the vector which contains cumulative sum for each thread starting from the beginning of the grid.
  - single_bit pointer to the vector which contains the single bit of the numbers.

- float **perform_milliseconds_to_seconds** (float elapsed_in_ms)
  Function to perform time from milliseconds to seconds.

  Parameters

  - elapsed_in_ms elapsed time in milliseconds.

- unsigned int **count_flops** (unsigned int number_of_all_blocks)
  Function to count flops.
  Parameters
  - number_of_all_blocks number of all grid's blocks into the GPU.

- float **perform_Mflop_per_sec** (float elapsed_in_ms, unsigned int number_of_all_blocks)
  Function to perform Mflops.
  Parameters
  - elapsed_in_ms elapsed time in milliseconds.
  - number_of_all_blocks number of all grid's blocks into the GPU.

- void **check_error_from** (cudaError_t cuda_error, bool show_result = true)
  Function to check error from CUDA operation.
  Parameters
  - cuda_error CUDA error.
  - show_result print eventual error into the console.

- void **radix_sort_on_GPU** (unsigned int *input_array, unsigned int *output_array_GPU, unsigned int *number_of_all_blocks, float *mflop_per_sec, float *elapsed_from_GPU)
  Function to perform RadixSort by using GPU.
  Parameters
  - input_array pointer to the vector to be sorted.
  - output_array_GPU pointer to the sorted vector.
  - number_of_all_blocks number of all grid's blocks into the GPU.
  - mflop_per_sec Mflops.
  - elapsed_from_GPU elapsed time committed from GPU.

- void **read_input_from_file** (unsigned int *array)
  Function to read an unsorted array from a file.
  Parameters
  - array pointer to the sorted vector.

- bool **file_exist** (char *filename)
  Function to check if file exists.
  Parameters
  - filename directory.

- void **make_csv_of** (unsigned int number_of_all_blocks, float mflop_per_sec, float elapsed_from_GPU, double elapsed_from_CPU)
  Function to make a csv of the measures.

  Parameters
  - number_of_all_blocks number of all grid's blocks into the GPU.
  - mflop_per_sec Mflops.
  - elapsed_from_GPU elapsed time committed from GPU.
  - elapsed_from_CPU elapsed time committed from CPU.

- unsigned int ***errors_in_comparison_between*** (unsigned int *output_array_CPU, unsignedint *output_array_GPU)
  Test function to detect errors between CPU and GPU.

  Parameters
  - output_array_CPU pointer to the sorted vector by CPU.
  - output_array_GPU pointer to the sorted vector by GPU.

- void ***compare_output_between_CPU_and_GPU*** (unsigned int *output_array_CPU, unsigned int *output_array_GPU)
  Test function to compare output between CPU and GPU.

  Parameters
  - output_array_CPU pointer to the sorted vector by CPU.
  - output_array_GPU pointer to the sorted vector by GPU.

How to run

Folders

- `*global*`: contains the script GLOBAL_MEMORY.cu;
- `*shared*`: contains the script SHARED_MEMORY.cu;
- `*texture*`: contains the script TEXTURE_MEMORY.cu;
- `*measures*`: contains the measures of global, shared and texture memory **(DO NOT DELETE `*measures*` FOLDER!!)**

Common files

These files have been made to generate measures and extract means of them:

- `*cuda_execute.bash*` □ bash script that runs .cu files 200 times
- `*cuda_means.py*` □ python script that calculates means of the measures
- `*random_numbers.txt*` □ text file contains random numbers to be sorted

How to run single program

Each version of this program is executed from the local GPU:

1. `nvcc .\name_of_program.cu -o .\name_of_program` to compile the program;
2. `./name_of_program` to execute the program.

How to run all programs

To run the bash file, after any change, **it is necessary previously to compile global, shared and texture version**:

1. `bash cuda_execute.bash`

So, to extract the mean of the measure and to print them on the terminal (paying attenction to the set correctly **memory_type_list** and **thread_per_block_list** parameters since they depend from the content of `measures` folder):

2. `python3 cuda_means.py`

Bibliography

The sequential part of the algorithm is taken from Internet at the link: https://www.programiz.com/dsa/radix-sort

The idea of CUDA algorithm is taken from Internet at the link: http://www.compsci.hunter.cuny.edu/~sweiss/course_materials/csci360/lecture_notes/radix_sort_cuda.cc

License